

---

Masters Theses

Student Theses and Dissertations

---

Fall 1989

## Speech synthesis using a digital modulation scheme on the IBM personal computer

John Glenn Wallace

Follow this and additional works at: [https://scholarsmine.mst.edu/masters\\_theses](https://scholarsmine.mst.edu/masters_theses)



Part of the [Electrical and Computer Engineering Commons](#)

Department:

---

### Recommended Citation

Wallace, John Glenn, "Speech synthesis using a digital modulation scheme on the IBM personal computer" (1989). *Masters Theses*. 4634.

[https://scholarsmine.mst.edu/masters\\_theses/4634](https://scholarsmine.mst.edu/masters_theses/4634)

This thesis is brought to you by Scholars' Mine, a service of the Missouri S&T Library and Learning Resources. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact [scholarsmine@mst.edu](mailto:scholarsmine@mst.edu).

SPEECH SYNTHESIS USING A  
DIGITAL MODULATION SCHEME ON THE IBM PERSONAL COMPUTER

BY

JOHN GLENN WALLACE II, 1961 -

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI- ROLLA

In Partial Fulfillment of the Requirements for the Degree

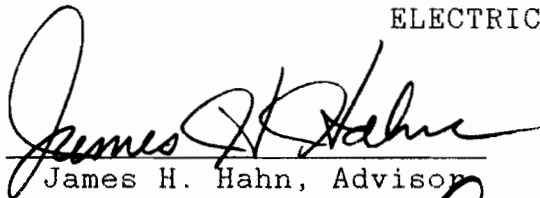
MASTER OF SCIENCE

in

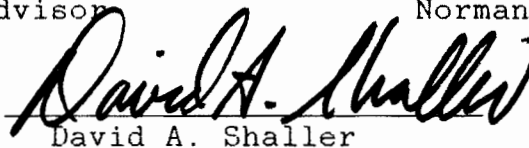
T5997  
Copy 1  
68 pages

ELECTRICAL ENGINEERING

1989

  
James H. Hahn, Advisor

  
Norman R. Cox

  
David A. Shaller

© 1989  
John Glenn Wallace II  
All Rights Reserved

## ABSTRACT

Previously the primary output interface that users of personal computers have had available to them were limited to the crt or the printer. This paper presents a method for implementing a voice synthesizer on an IBM personal computer using only the hardware already contained in the personal computer. Presented is a modulation method which can be used with the IBM PC to obtain good quality voice output. The method consists of 3 parts, 1) sampling and recording the voice, 2) processing and filtering these samples and 3) using these processed samples to drive the speaker. A description of the hardware used to sample the speech and flowcharts for all software are included.

## ACKNOWLEDGEMENT

I would like to express my thanks and appreciation to Dr. James Hahn for his guidance, suggestions and patience he showed me while this thesis was being prepared. I would also like to thank Dr. Norman Cox and Dr. David Shaller for serving as committee members.

Thanks also go to my friends and coworkers who listened patiently as I presented my ideas to them. Special thanks also go to all the volunteers who participated in the comparison testing.

Most of all, I would like to thank my family. My wife, my child and my parents have given me great encouragement in the pursuit of my education. My family has been very understanding through all of the long hours spent away from them and I owe them a great deal.

## TABLE OF CONTENTS

	page
ABSTRACT . . . . .	iii
ACKNOWLEDGEMENT . . . . .	iv
LIST OF ILLUSTRATIONS . . . . .	vi
I. INTRODUCTION . . . . .	1
II. REVIEW OF CURRENT LITERATURE . . . . .	3
III. FUNDAMENTALS OF SPEECH PRODUCTION . . . . .	5
IV. SPEECH SYNTHESIS . . . . .	7
V. COMPUTER DESCRIPTION . . . . .	9
VI. IMPLEMENTATION . . . . .	12
VII. DISCUSSION . . . . .	26
VIII. COMPARISON TESTING . . . . .	36
IX. CONCLUSIONS . . . . .	38
BIBLIOGRAPHY . . . . .	39
VITA . . . . .	41
APPENDIX A- DATACAP FLOWCHART . . . . .	42
APPENDIX B- SPEAK INTERRUPT TIMING . . . . .	48
APPENDIX C- SPEAK FLOWCHART . . . . .	51
APPENDIX D- PREPRO FLOWCHART . . . . .	56
APPENDIX E- PHONEME SENTENCES . . . . .	61

## LIST OF ILLUSTRATIONS

Figure 1- PCM TO PWM CONVERSION . . . . .	20
Figure 2- SPEECH SYNTHESIZER BLOCK DIAGRAM . . . . .	24
Figure 3- PC FREQUENCY RESPONSE . . . . .	28
Figure 4- SPEECH SEGMENT . . . . .	30
Figure 5- SPEECH SPECTRUM . . . . .	30
Figure 6- SPEECH SEGMENT, SCALING ONLY . . . . .	32
Figure 7- SPEECH SPECTRUM, SCALING ONLY . . . . .	32
Figure 8- SPEECH SEGMENT, SCALING AND PEAK CLIPPING . .	34
Figure 9- SPEECH SPECTRUM, SCALING AND PEAK CLIPPING .	34

## I. INTRODUCTION

The increasing popularity of computers and software has brought with it major advances in the man/machine interface. Just as the keyboard replaced the card reader as the main input device on computers, it is now being replaced with devices which make the interface to the computer as natural for a user as possible. The mouse, the light pen, the track ball, the digitizing tablet and even voice recognition systems are now being used to allow the user to enter data into the computer as easily as possible.

Despite the advances in the input side of the interface the primary output interface is still a cathode ray tube (CRT) or printer. While these work fine in many situations, various circumstances exist where an interface which is more "user friendly" is needed. An example of a situation where a better output interface might be needed would be a computer being used by a sightless individual. While this individual could use the keyboard to enter data or programs it would be desirable for the user to be able to have the output from the computer generated in such a way as to be useful to him or her. Another example might be educational software whose purpose is to teach a user a foreign language. While the software can convey grammar rules and definitions using only a CRT, a better interface is needed to convey the inflections and pronunciation of the language.



These are examples where a speech output interface from the computer could be used much more effectively than the CRT or the printer. The speech output interface could provide the user with a very natural output method which would be much more effective than the CRT or the printer.

This paper describes a method of generating speech output from a personal computer (PC). Section II reviews the literature and comments on current work in related fields. Section III describes the fundamentals of speech processing and shows how they relate to the problem at hand. Section IV details synthetic speech synthesis methods and describes the method chosen for this project. Section V describes the computer/speaker interface. Section VI describes the implementation of the synthesizer. Section VII provides a discussions of the problems which were encountered. Section VIII describes the results of a comparison test used to compare this synthesizer with another PC based synthesizer. Finally, section IX offers conclusions and suggestions for further work.

It should be noted that this project was developed using an IBM PS 2 Model 50. While the synthesizer is not limited to running on a Model 50 it does require the computer to have approximately the processing speed equivalent to an IBM PC AT or greater. It will also run on PC's which are 100% compatible with the IBM machines.

## II. REVIEW OF CURRENT LITERATURE

In recent years, it has been a general trend to provide voice output in many areas where a user must interface with a computer. The Texas Instruments Speak & Spell educational toy was one of the first consumer products which provided a speech synthesizer output and was available to the general public [1]. Many consumer products now have some type of speech output capability. Cars, appliances, and soda machines are examples of a product whose operation has been enhanced by a speech synthesizer output.

While future PC's will most likely incorporate some type of speech input/output (I/O) [2], there are many PCs currently in use whose value could be increased if some type of speech interface could be developed for them. While work has been done in the area of PC based speech synthesis, this work has generally been limited to add-in boards or peripherals which have special applications software which control them [3]. The requirement to purchase and install an add-in board or peripheral to add speech synthesis to the PC limits the potential applications of these products. Many users would choose not to install these add-in products. This would leave software developers in an uncertain market environment and there might be some reluctance to develop the necessary applications software to drive these products.

In order to present the advantages of speech based output to the largest possible group of users, a speech synthesizer

needs to be developed which will deliver good quality speech from the PC, but will not require any add-in boards or peripherals.

Some work has apparently been done in this area, although no documentation on the work could be found in the literature. A program (obtained through the public domain) which does attempt to provide speech output on the PC using only the PC's internal hardware does exist. The software appears to use a type of phoneme synthesis and the resulting quality of the speech is somewhat unintelligible. Section VII will detail the results of a comparison of the quality of the speech from the public domain software with the speech generated by the method outlined in this paper.

### III. FUNDAMENTALS OF SPEECH PRODUCTION

Speech is generated by the actions of the lungs, vocal tract, mouth cavity, nasal cavity, lips and tongue working together. A sound is started by blowing puffs of air across the vocal cords. This action causes the vocal cords to vibrate at a frequency which determines the pitch that will be associated with the sound. The vibration of the vocal cords generates pulses which then excite the mouth and nasal cavity. The lips, tongue, mouth and nasal cavity then work together to form acoustic resonances which provide the sound with a distinguishing characteristic [4].

Two basic types of sounds can be produced by the human vocal system. These are called voiced and unvoiced. A voiced sound is produced when the pulses of air are generated in a periodic fashion. The voiced sound has a constant pitch (which is the period of the air pulses) and a definite structure of frequency peaks (called formants) due to the resonances of the vocal tract. An example of a voiced sound is the letter "A". Unvoiced sounds occur when the vocal tract constricts the flow of air. The result is a random noise with no definite pitch structure. An example of a unvoiced sound is the letter "S" [5].

Normal speech contains frequencies from about 50 Hz to about 10 KHz. It is known that most of the intelligence information in a speech waveform is contained in the frequencies below about 4 KHz. The frequencies above 4 kHz

and below 10 khz contain the information which contributes to the quality of the speech. It is generally accepted that a sampling rate of 8 Khz is sufficient for telephone toll quality speech and this sampling rate will be used for this project [6].

#### IV. SPEECH SYNTHESIS

There are two major techniques used in speech synthesis. The first technique is called waveform coding. In this method a speech waveform is sampled at its Nyquist frequency and the waveform can be reproduced by direct conversion of these samples back to an analog signal with the aid of a Digital to Analog (D/A) converter. The second method is called the Analysis/Synthesis method. In this method samples are again taken at the Nyquist frequency. Instead of storing the samples, analysis routines are performed on the sampled data to extract certain parameters. These parameters are then used to create a model of the vocal tract. The vocal tract model can then be used to reconstruct an approximation to the original waveform [5, p.346].

As one would expect, there are various tradeoffs associated with using either of these methods. The waveform method has the best quality and is simplest to implement, but takes up a lot of memory for storage of the samples. The analysis method takes up less memory but the quality is not as good as that obtainable using the waveform method. There were two major criteria which influenced the decision as to which method would be chosen for this speech synthesizer. The first criteria was the speech quality desired. Waveform coding inherently has better quality than the analysis/synthesis method so it meets the first criteria.

The second criteria is the processing time required for the reconstruction of the speech. Since the Central Processing Unit (CPU) time must be shared between the speech synthesizer and other programs, a premium is placed on the CPU time that is available for the speech synthesizer to use. Waveform coding requires less "real time" CPU processing time since no calculations are required for each output sample.

Based upon the above criteria, it was decided to use a waveform coding technique for this speech synthesizer.

## V. COMPUTER DESCRIPTION

The personal computer used to develop the speech synthesizer for this project was an IBM Personal System 2-Model 50. The Model 50 has an 80286 microprocessor which runs at a clock speed of 10 Mhz. The Read Only Memory (ROM) Basic Input Output System (BIOS) calls are compatible with the older IBM PC XT and AT models and the Model 50 can generally run all the software written for these models.

The synthesizer was designed so it could be used on any IBM PC or compatible with sufficient processing speed. The specific limitations will be discussed in Section VI (Implementation).

Unfortunately, the sound generation system in the PC was not designed with speech synthesis in mind. A permanent magnet type of speaker is mounted inside the PC cabinet. The speaker is driven by the output of a linear amplifier. The output of the driver is low pass filtered to remove the high frequency edges from the driver signals before being passed to the speaker. No form of amplitude modulation of the speaker is available.

There are two input sources which can drive the speaker. One is the output of an 8255 programmable peripheral interface (PPI) chip. The other is one channel of an 8253 programmable timer chip. Both of these devices are controlled by using software to write to an output port [7].



The PPI can be used to generate a square wave output to the speaker by successively turning the speaker bit on the 8255 on and off. This is the general method used to generate tones on the PC. The problem with using the 8255 to drive the speaker is that it is very CPU intensive since the CPU must make 2 port writes per cycle of the square wave (one to turn the bit on, another to turn the bit off). In instances where the CPU cannot devote this much time to driving the speaker the 8253 can be used. The timer chip can be set up to generate a square wave by writing one number to it. It will continue generating this square wave until it is turned off. This frees the CPU for other duties while still providing a tone on the speaker.

The computer also has one channel of the 8253 set up to drive an interrupt line on the CPU. This interrupt occurs at a fixed rate (18.2 Hz) and provides the CPU with a link to real time. The BIOS of the computer uses this interrupt to service the keyboard, disk drives and the time of day clock [8].

This interrupt is invisible to a user's program since it is wired in as a hardware interrupt. Each time this interrupt occurs the CPU stores the current state of the CPU then jumps to an interrupt routine stored in the BIOS. The interrupt routine takes care of the computer housekeeping functions mentioned above and then restores the CPU to its previous state. Control is then returned to the application software

which was interrupted. In this way the operating system is able to do its job without any interference to the application software.

## VI. IMPLEMENTATION

In order to implement the speech synthesizer using the waveform coding method, speech samples of different words were taken. These speech samples were then processed to form output samples and the output samples were stored on disk. To run the speech synthesizer, the user must provide a filename to the synthesizer. The associated file should contain the output samples which represent the word which is to be spoken. The synthesizer then reads the data stored in the file and applies a modulation method to the data to drive the speaker.

The hardware used to sample the speech signals consisted of a Radio Shack part number (P/N) 33-1089 microphone, a wirewrap board which provided an antialiasing filter and amplification for the speech signal and an Interface Technologies Analog I/O board for the IBM PC.

The microphone output was amplified to a level compatible with the Analog I/O board by a 3 stage amplifier which used an National Semiconductor P/N LM324 op amp. The gain of the last stage was adjustable so the output signal amplitude could be precisely matched to the +/- 5 volt input range of the Analog I/O board. The amplifiers were designed to have a flat frequency response to greater than 50 khz so as not to degrade the signal.

The antialiasing filter was a low pass filter with a cutoff frequency of approximately 3.7 Khz. In order to

facilitate changes in the frequency response of this filter (if needed) it was designed using a National Semiconductor switched capacitor filter, P/N MF10. The cutoff frequency of the MF10 can be varied by changing the clock frequency to the MF10 [9]. A 12 Mhz oscillator was used to generate the basic system clock. A Signetics 14 stage binary ripple counter, P/N 74HCT4060, was then used to divide the clock down to the appropriate rate [10]. The clock rate to the MF10, and hence the cutoff frequency of the antialiasing filter, can be changed by simply changing the stage where the clock is taken from on the 74HCT4060.

The Analog I/O board used was an Interface Technologies ML16 board. The ML16 board is a basic Analog/Discrete interface board for the IBM PC XT or AT. It occupies one slot in the PC and is resident on the PC bus. The board has a multi channel analog to digital converter, a digital to analog converter and 8 discrete inputs and 8 discrete outputs. The board is very flexible and can be set up to use an on board 8253 to generate a pacer clock which can be used to trigger an Analog to Digital (A/D) conversion and also generate a hardware interrupt to the PC's CPU. The data sample can then be read through a port on the ML16 board. Reading this port also readies the board for conversion of another sample [11].

An assembly language program (DATACAP.ASM) was written to initialize the ML16 board and to service the conversion interrupts. The program configured the ML16 board for 5 volt

bipolar input, straight binary coding and set an A/D conversion rate of 8 Khz.

After initialization and installation of the interrupt handler, the program prompts the user for a keyboard input. When the user responds by pressing a key the program starts the ML16 board's A/D conversion process by controlling the gate input (which is externally tied to output discrete 0) to the pacer clock's 8253 generator. The program then samples and stores 2 seconds (16000 samples) of data. When 16,000 samples have been taken the ML16's pacer clock is turned off the conversion interrupt routine is removed and the data that has been stored in an array is now written to floppy disk. For a more complete explanation of the DATACAP program a flow chart is included in Appendix A.

With the speech samples stored on disk, a modulation method had to be found which could be applied to these samples and which would generate a suitable output using the PC's limited sound generation capabilities.

Since the ultimate use of this synthesizer is to be used with an application program, one of the requirements was that the synthesizer should use as little of the CPU's time as possible. It was decided to use the speaker driver 8253 to drive the speaker since it requires far less CPU intervention to control the speaker than the PPI does.

A pacer clock was also needed in the PC in order to send a data sample to the speaker driver 8253 at the proper time. The criterion for the pacer clock was that it had to be

connected to one of the hardware interrupts on the CPU so the software could send the samples out to the 8253 without having to resort to a polling arrangement. The only clock available in the PC which satisfied these requirements was the hardware system timer referenced in Section V. The only problem with using this timer was the fact that it is set by the PC BIOS to operate at an 18.2 Hz rate and it is necessary to maintain this rate in order to keep the PC DOS from losing internal sync and locking up the computer.

The system timer is generated by an 8253 which the BIOS initializes to generate a pulse output at the 18.2 Hz rate. It was found that the frequency of this interrupt could be increased to 8 KHz by reprogramming the 8253 to generate a pulse output at a rate of 8 KHz. In order to do this, the interrupt is first masked so no interrupts can occur. Next, a new interrupt routine is installed in place of the previous routine. Finally, the new count value is written to the 8253 and the interrupt is then unmasked. The new interrupt routine will now be called at a rate of 8000 times per second. In order to maintain the Disk Operating System's (DOS) internal sync the new interrupt routine must turn control over to the old interrupt routine at a rate which equals the 18.2 Hz rate. To insure that this is done, the interrupt routine keeps a count of how many times it has been executed since the last time it called the DOS interrupt routine. When this count is equal to  $8000/18.2$  (440), the count is cleared and control is passed to the DOS interrupt routine. This keeps the DOS

running correctly and provides a pacer clock which can be used to output samples to the speaker driver.

In order to use the above mentioned pacer clock to send data samples to the speaker driver an assembly language program (SPEAK.ASM) was written. The program consists of 2 basic parts. The first part contains the initialization routines which set up the system timer and install the interrupt routine. The second part consists of the interrupt routine itself.

The initialization routine first loads the data to be output to the speaker into a data array in Random Access Memory (RAM) from disk storage. It then reprograms the 8253 to generate the timer interrupt at the much faster rate. After this has been done, the program waits until the total number of data samples (encoded in the data which was loaded from disk) have been sent to the speaker driver. When this happens, the program restores the system timer to the original status and reinstalls the BIOS system timer interrupt routine. The program then terminates itself.

When the interrupt handler is entered, it first increments the index on the data array so that the next sample will be sent to the speaker. It then obtains this sample and transfers its value to the speaker driver 8253. It then increments a counter and checks to see if the counter value is equal to 440. If so, it resets the counter value and calls the DOS system timer procedure. If the value is not 440, the interrupt procedure returns. It should be noted that this

interrupt procedure has been optimized somewhat for speed. Since this procedure is being executed 8000 times per second, it must run as fast as possible. If the interrupt routine takes too long to execute, it could mask out the next interrupt which occurs and some of the data samples would be lost. This requirement limits the use of this speech synthesizer on computers which do not have sufficient speed to keep up with these requirements. Appendix B details the time required for the interrupt routine to execute and arrives at a lower limit for the clock speed of the 80286 CPU. See Appendix C for a flowchart of the SPEAK program.

The problem of getting data into the system and getting data out to the speaker has been addressed, but there is one very important area which must still be addressed, modulation of the data samples to obtain a speech waveform at the speaker. The fact that the speaker is driven by a digital signal precluded the use of any type of amplitude modulation so it was decided to use a form of digital modulation.

The speaker driver 8253 can be set up in a mode (defined as mode 0) where the output will initially be low. The output will remain low until the terminal count, which has been previously loaded in, is reached. The output will then go high until the timer is reset [12].

By using this mode of the 8253 along with the system timer as the pacer clock, the SPEAK software can drive the speaker with a waveform which is low for an arbitrary time and then goes high until the next edge of the pacer clock. The



length of time the output is to stay low is controlled by the value loaded to the 8253 at each occurrence of the pacer clock. After a value is loaded to the 8253, the output from the speaker driver will remain low until the terminal count is reached at which point the output will go high. The output will stay high until the next system timer interrupt occurs, and a new value is loaded into the 8253 speaker driver. A comparison of this waveform with the Pulse Width Modulated Waveform (PWM) described by Ziemer and Tranter [13] shows that they are identical except for an unimportant polarity reversal.

The conversion from the Pulse Code Modulated (PCM) input waveform to the PWM output waveform is accomplished in three steps. First, a DC offset is added to the PCM encoded input samples to make all the samples positive. Next these samples are scaled to insure that the maximum amplitude sample generates a pulse which has a maximum pulse width. This is accomplished by the following equation:

$$y[n] = \text{RND} (x[n] * A) \quad (1)$$

where  $y[n]$  represents the output samples, the function RND represents rounding the floating point multiplication to the nearest integer and  $A$  represents a scale factor chosen to make the maximum value of  $y[n]$  correspond to a maximum pulse width. The final step in the conversion process occurs when the output samples are loaded into the 8253. The 8253 then generates a pulse output whose width is proportional to the

value loaded into it. Figure 1 shows an illustration of the conversion of the PCM input samples to a PWM output.

The scale factor in (1) is chosen so that when the maximum input value is multiplied by the scale factor the output pulse width will be at a maximum. The value of the scale factor then, is just the ratio of the maximum value the output can be to the maximum value the input can be. To calculate the maximum output value, the ratio of the sampling period,  $T_{\text{samp}}$ , to the period of the clock input to the 8253,  $t_{\text{clock}}$  is taken. This maximum output value represents the largest value which can be loaded to the 8253 without causing interference with the adjacent output pulses. Given the clock input frequency to the 8253 speaker driver (1.19 Mhz- A constant on all PC's) and the period of the sampling rate (125  $\mu\text{sec}$ ), the maximum value which the output samples can have is:

$$\alpha = \frac{T_{\text{samp}}}{t_{\text{clock}}} = \frac{125 \mu\text{sec}}{0.84 \mu\text{sec}} \approx 148 \quad (2)$$

The value of the scale factor is then:

$$A = \frac{148}{255} \approx 0.58 \quad (3)$$

The spectrum of a PWM signal has been shown to contain a component present at the input frequency,  $f_m$ , and intermodulation products of the input frequency and the sampling frequency defined by  $m \cdot f_s + n \cdot f_m$  and  $m \cdot f_s - n \cdot f_m$  for all  $m, n > 1$ . Recovery of the input frequency occurs by filtering the PWM to remove the intermodulation frequencies.

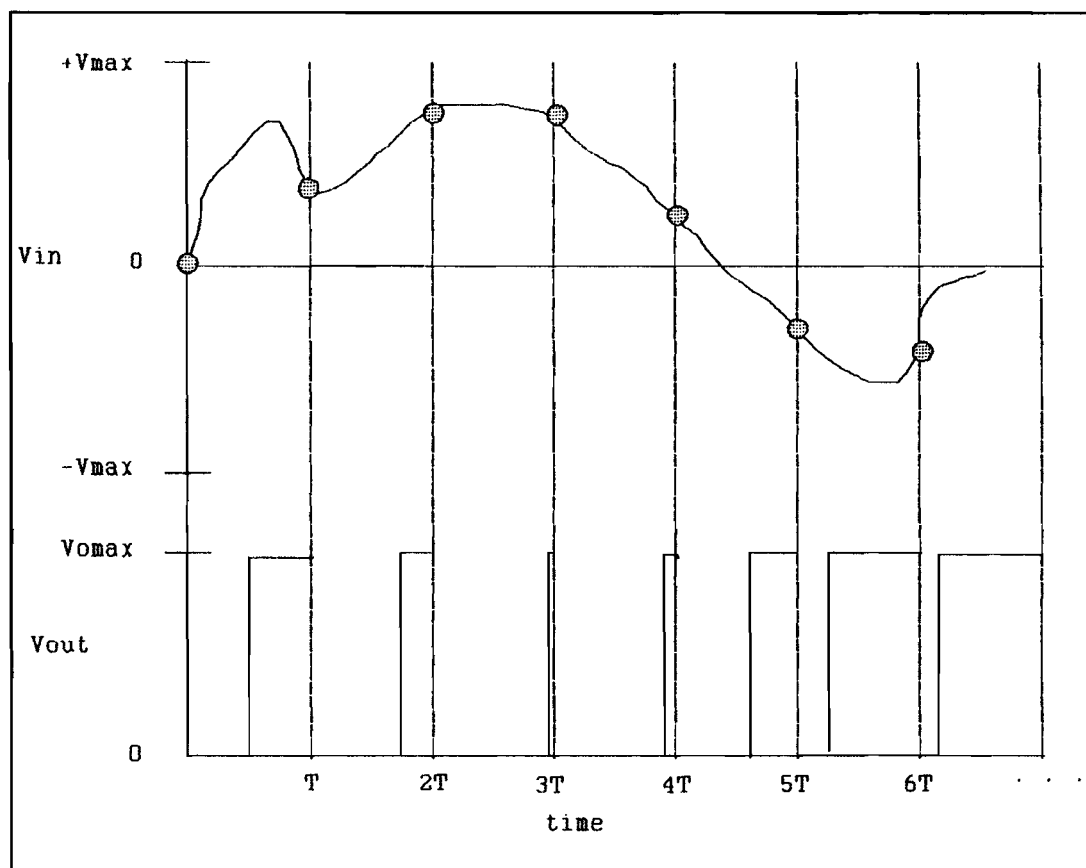


Figure 1- PCM TO PWM CONVERSION

If the sampling rate is high enough and the cut off frequency is chosen correctly, the input signal can be recovered by low pass filtering the PWM signal [14, p. 175]. As described in Section III, the output from the speaker driver is low pass filtered before being sent to the speaker. Assuming, then, that the speakers bandpass is low enough, the speaker filter will act as the demodulator for the PWM signal. In order to provide the highest quality speech output, the input samples must be processed before they are sent to the 8253 speaker driver to form the PWM waveform. A program was written which takes the input samples, does some processing on them, and generates an output file. This output file becomes the input file for the speak program. This preprocessing program is called PREPRO.

PREPRO first loads the data from disk. The input data is in the form of an integer whose range is 0 to 255. Next, the DC component is removed by subtracting the average from each sample. This operation involves splitting the samples into 100 segments with each segment consisting of 80 samples. The average value for each segment is then calculated. This results in 100 average values. These 100 average values are then averaged together. This value represents the DC component of the input data and it is removed from each sample. It is necessary to calculate the average in this way to prevent integer overflow when doing the calculations.

After the DC component is removed from the samples the program removes the front end silence and the back end silence

periods. This is necessary since the sampled speech data will almost always have some period of silence captured before the actual data starts. In order to remove this silence period the program looks for the first sample in the data array which is some percentage of the maximum sample amplitude. It then chops off all the data samples which occur before this point. The percentage figure is entered by the user and is arrived at by repeated trials using an increasing percentage of the maximum sample amplitude.

In order to remove the silence from the end of the speech data, the total energy of all the sampled data (less the front end silence) is calculated. Since the samples taken during the silence periods are relatively low amplitude noise, the speech data will be the major contributor to the total energy. Once the total energy has been calculated, the program then calculates the total number of samples required to retain a certain percentage of the total energy. This number then gives the total number of speech samples needed to represent the speech waveform. The percentage of the energy to retain is supplied by the user and is trimmed by trial and error. For the samples which were taken it was found that a value between 70 and 85 % usually worked best.

After the silence periods have been removed the input data is converted to PWM output by implementing (1). A new file is then created and the output data is written into this file. It is this file which becomes the input to the SPEAK

program. See Appendix D for a flow chart of the PREPRO program.

A summary of the steps involved in going from the sampling of speech data to the generation of speech data on the PC is as follows:

- 1) Sample speech using an ML16 board, filter hardware and DATACAP software.
- 2) Process the speech samples with the PREPRO program.
- 3) Run the SPEAK program supplying the output of the PREPRO program as input.

These steps are shown in block diagram form in Figure 2.

To obtain a rough idea of the quality this speech synthesizer is capable of, the signal to noise ratio (SNR) can be calculated assuming the quantization of the speech signal is the only noise introduced into the channel. The input quantization is due to the A/D conversion process and has a step size of  $(256)^{-1}$  which is 8 bits. The output quantization is due to the PCM to PWM conversion process and has a step size of  $(148)^{-1}$ . The SNR can be calculated by the following method [14, pp. 297-300]:

$$\text{SNR} = 10 \log\left(\frac{\sigma^2}{D}\right) = 10 \log\left(\frac{\sigma^2}{D_1 + D_2}\right) \quad (4)$$

Where  $\sigma$  is the input signal standard deviation,  $D$  is the total distortion,  $D_1$  is the distortion from the first quantizer and  $D_2$  is the distortion from the second quantizer.

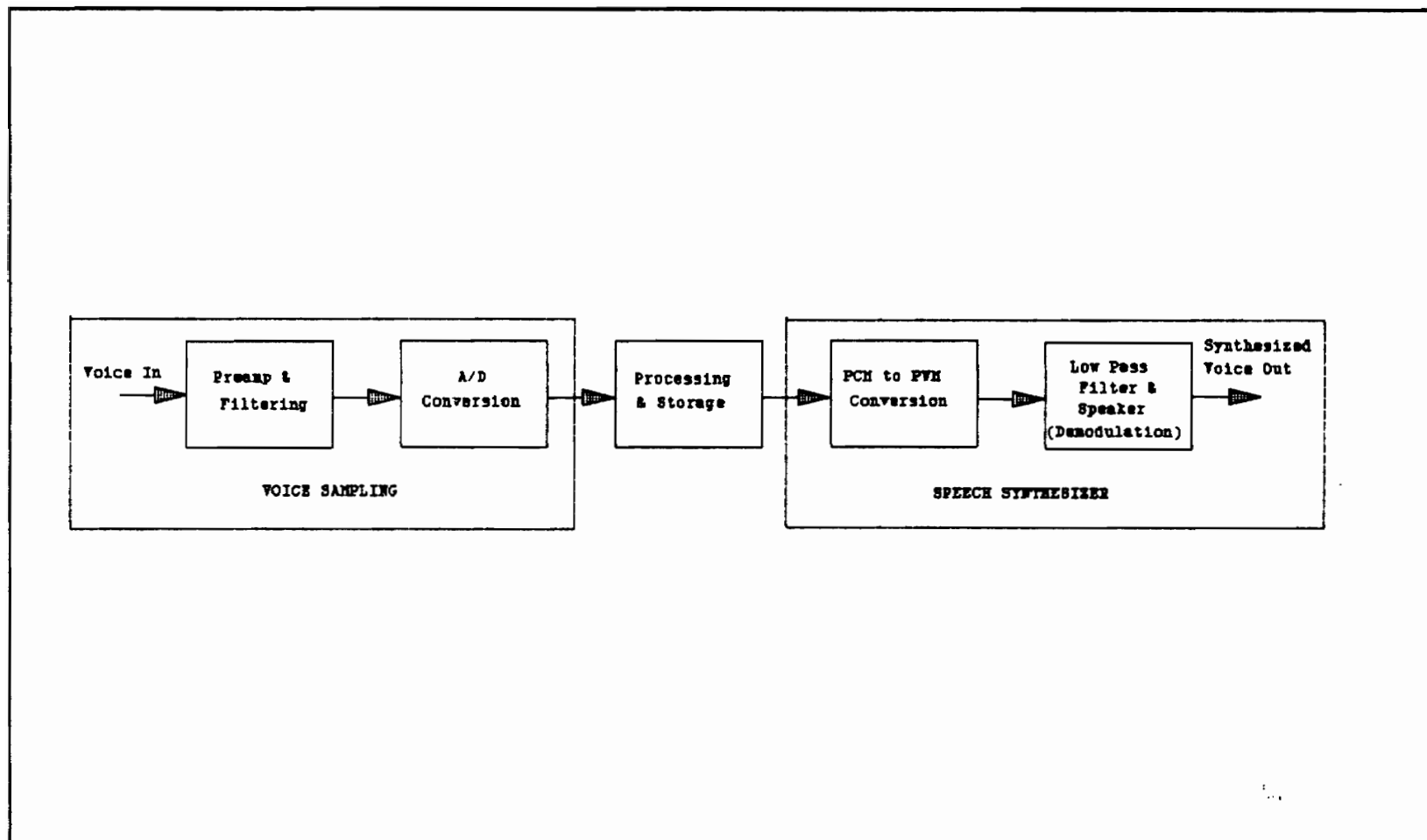


Figure 2- SPEECH SYNTHESIZER BLOCK DIAGRAM

Assuming that  $\sigma = 0.25$ , which is generally the case for speech signals, and using the formula

$$D = \frac{\delta^2}{12} \quad (5)$$

( $\delta$  is the step size) to calculate the distortions, the total SNR  $\approx 40$  db [14]. This is well above the 30 db SNR lower limit typically used for toll quality systems [6].

The preceding calculations show that the SNR of the system appears relatively high, however we will see in the next section that other factors besides quantization affect the system SNR and can lead to an excessive amount of noise in the output.



## VII. DISCUSSION

When the speech synthesizer was implemented as described in section VI, A high pitched noise was present along with the voice. To find the source of this noise it is helpful to analyze the equation for PWM signals. The equation, assuming the modulating signal is a sine wave with a frequency  $f_m$ , is as follows [13, p. 173]:

$$\begin{aligned}
 x(t) = & \frac{1}{2} h - \frac{1}{2} h \beta \cos(2\pi f_m t) + \\
 & \frac{1}{\pi} h \sum_m [ (-1)^m - J_0(m\pi\beta) ] \frac{1}{m} \sin(2\pi f_s t) - \\
 & \frac{1}{\pi} h \sum_m \sum_n \frac{1}{m} J_0(m\pi\beta) \{ \sin[2\pi(mf_s + nf_m)t + \frac{n\pi}{2}] + \\
 & \sin[2\pi(mf_s - nf_m)t + \frac{n\pi}{2}] \} \quad (6)
 \end{aligned}$$

Where  $h$  is the amplitude of the PWM signal,  $\beta$  is the modulation index and  $J_0$  is the Bessel function of the first kind of order 0.

The first term is a dc term and since the speaker is AC coupled it is not important. The second term is the signal component. The third term is a component at the sampling frequency. The fourth term represents the intermodulation products of the sampling frequency and the signal frequency.

It was apparent that the noise was caused by the sampling frequency and the intermodulation products. This implied that the PC speaker drivers' low pass filter did not have a sufficiently low cutoff frequency to properly reconstruct the

signal information from the PWM waveform. The frequency response of the speaker system was measured and the result is shown in Figure 3. Note that the cutoff frequency is approximately 5 KHz which is above the 4 KHz required for PWM.

Two possible solutions were considered to improve the way the system sounded. The first solution was to increase the sampling rate to a frequency at which the PC's low pass filtering was sufficient to recover the signal information without any added noise from the sampling rate. The other solution was to process the data in such a way as to make the amplitude of the signal frequency as large as possible while minimizing the amplitude of the noise frequencies.

Increasing the sampling rate would not have added any more complexity to the system and would have been easy to implement. There are two drawbacks with this solution, however. The first drawback is that an increased sampling rate, say 10 KHz, would have required more data to be stored for each word. For a 200 word vocabulary this would have been about 400 Kbytes of additional storage. For some systems this might not be acceptable. The second drawback is that an increase in sampling rate would mean that the processor interrupt rate for the pacer clock must be increased. This would place additional limitations on the type of machine which could run the speech synthesizer.

For the above reasons it was decided that the best solution would be to process the input data in such a way as

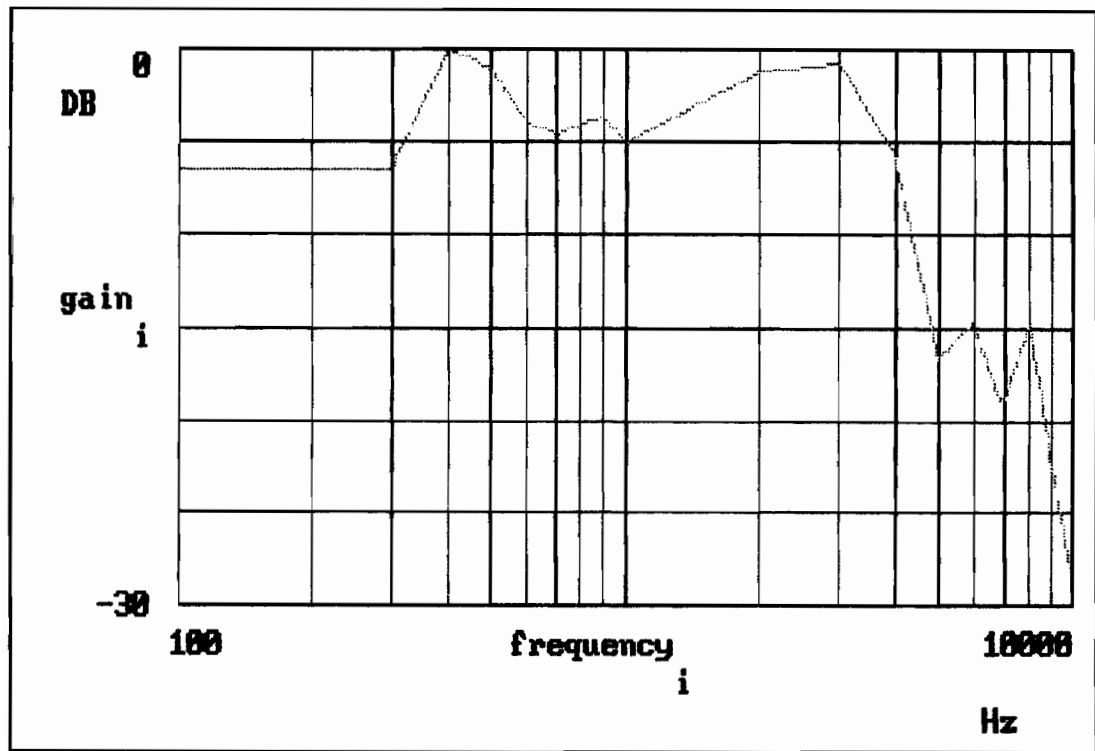


Figure 3- PC FREQUENCY RESPONSE

to amplify the amplitude of the signal information while suppressing the noise as much as possible.

To do this, it is necessary to return to (6). As can be seen, the amplitude of the signal is proportional to the modulation index,  $\beta$ .  $\beta$  is defined as having a value of 1.0 if the pulse width is as large as it can get without spilling over into the next pulse. Note that in the second term (sampling frequency component),  $\beta$  appears as an argument to the Bessel function. For this term as  $m$  increases  $J_0(m\pi\beta)$  will decrease. As  $m$  gets large this term will have no effect and the summation will go to  $+1-1+1-1$ , ad infinitum. The same term is also present in the intermodulation frequency components, but as a direct multiplier.

To gain a better understanding of the problem at hand, it is helpful to look at a segment of a speech waveform after it has been preprocessed. Such a waveform is shown in Figure 4. Note that the value that the signal can assume extends from 0 to 148, with a value of 148 being equivalent to  $\beta = 1$ . In order to determine the spectral response of the PWM waveform, a set of data samples, which represented a discrete time sampling of the PWM waveform, were calculated. These data samples were then passed as input to a program which calculated and plotted the spectral response of the input data samples. Figure 5 shows the spectral components of the waveform as calculated. As can be seen, there is a large component (relative to the signal information) which is located approximately at the sampling frequency. Due to the

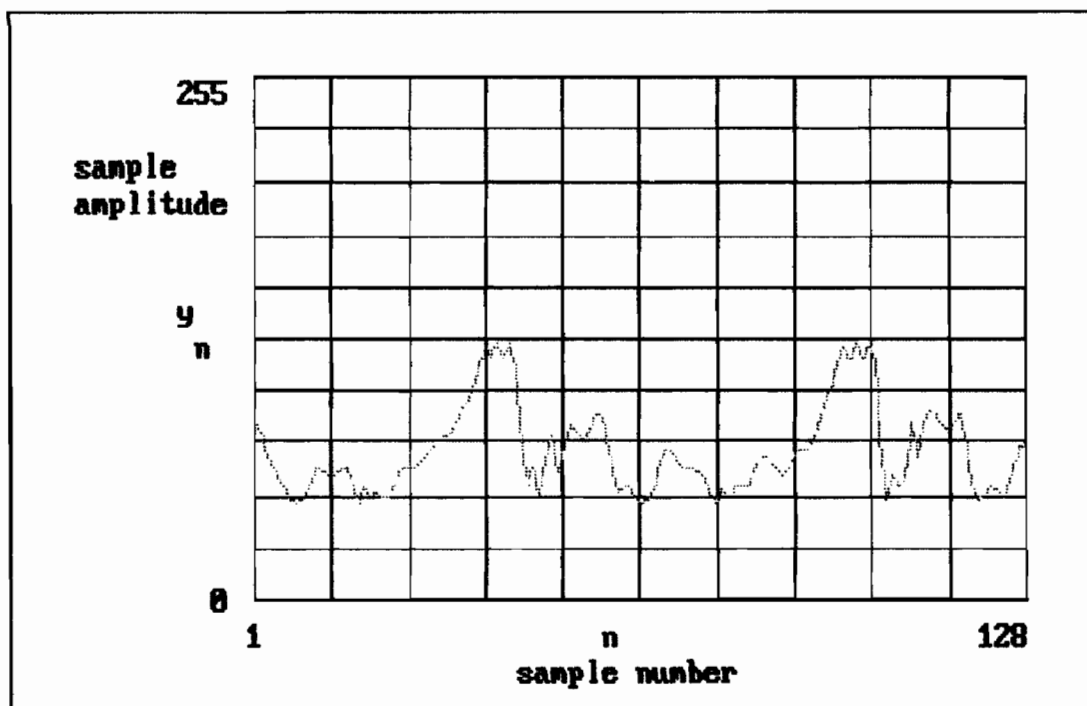


Figure 4- SPEECH SEGMENT

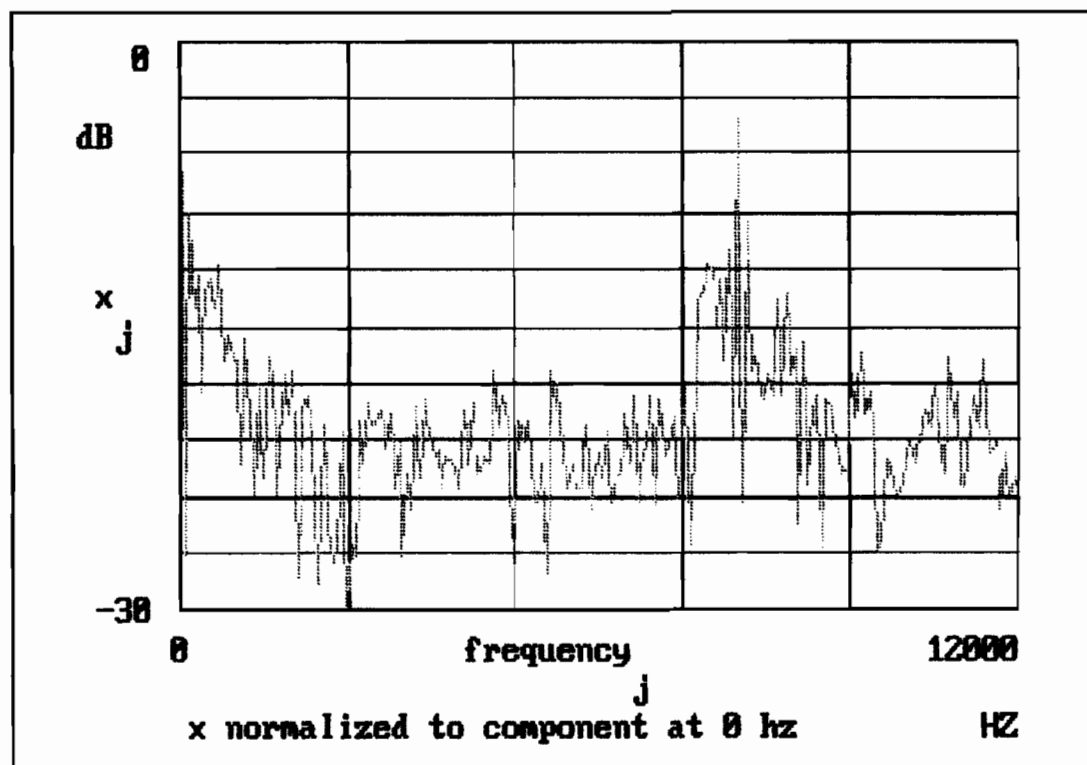


Figure 5- SPEECH SPECTRUM

high cutoff of the low pass filter, some of the energy from this component will leak through and appear at the speaker output.

Note that the signal range of the segment does not extend from 0 to 148, but rather from 50 to 125. In order to keep  $\beta$  large, the signal needs to fill as much of the range of 0 to 148 as possible. To ensure that this happens, a gain was applied to the signal in the preprocessor. The gain depended on the signal peak and was calculated to make the signal peak equal to the maximum output value allowed. The results of this gain operation are shown in Figure 6. As can be seen, the signal now occupies more of the range with Figure 7 showing the corresponding decrease in the amplitude of the noise in the frequency spectrum. While this reduction made some improvement in the way the system sounded, the presence of the noise was still loud enough to cause some problems understanding the speech output.

In order to further improve the sound quality, it was necessary to determine exactly what  $\beta$  means in this situation. Each output sample has a individual  $\beta$  defined for it. The samples which are at the maximum have a  $\beta$  of 1 (the largest). A sample value of 0 would have a  $\beta$  of 0. By looking at (6), it can be seen that samples that have a  $\beta$  at or near 1 drive the signal component up while driving the noise component down. It would be desirable to find a way to make as many of the samples as possible have a  $\beta = 1$ . This is equivalent to

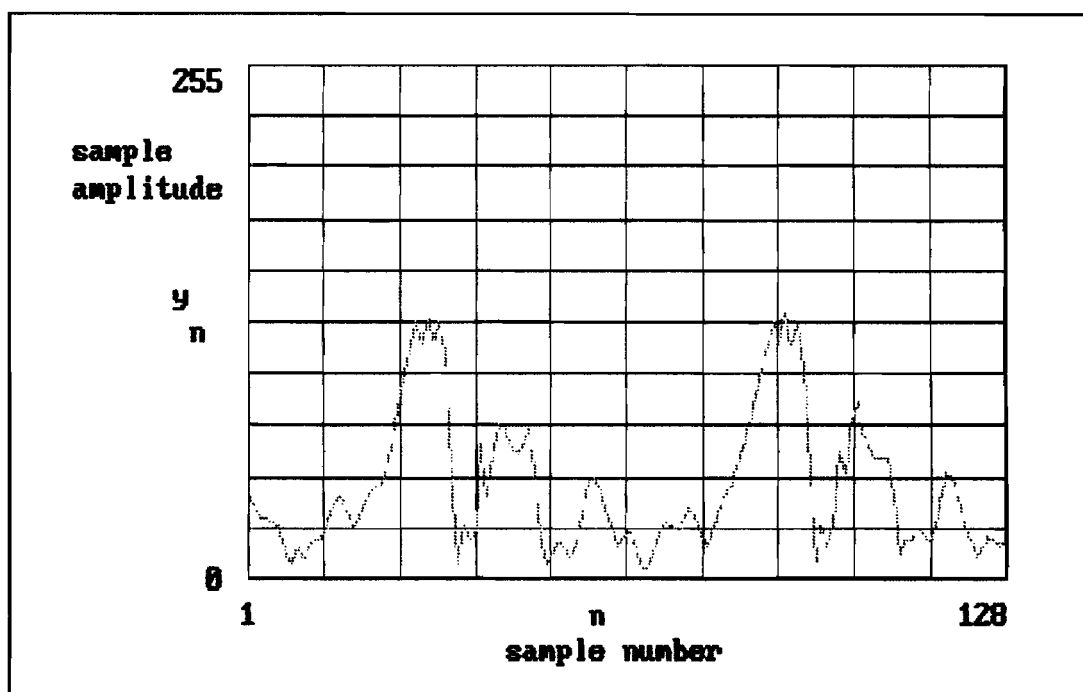


Figure 6- SPEECH SEGMENT, SCALING ONLY

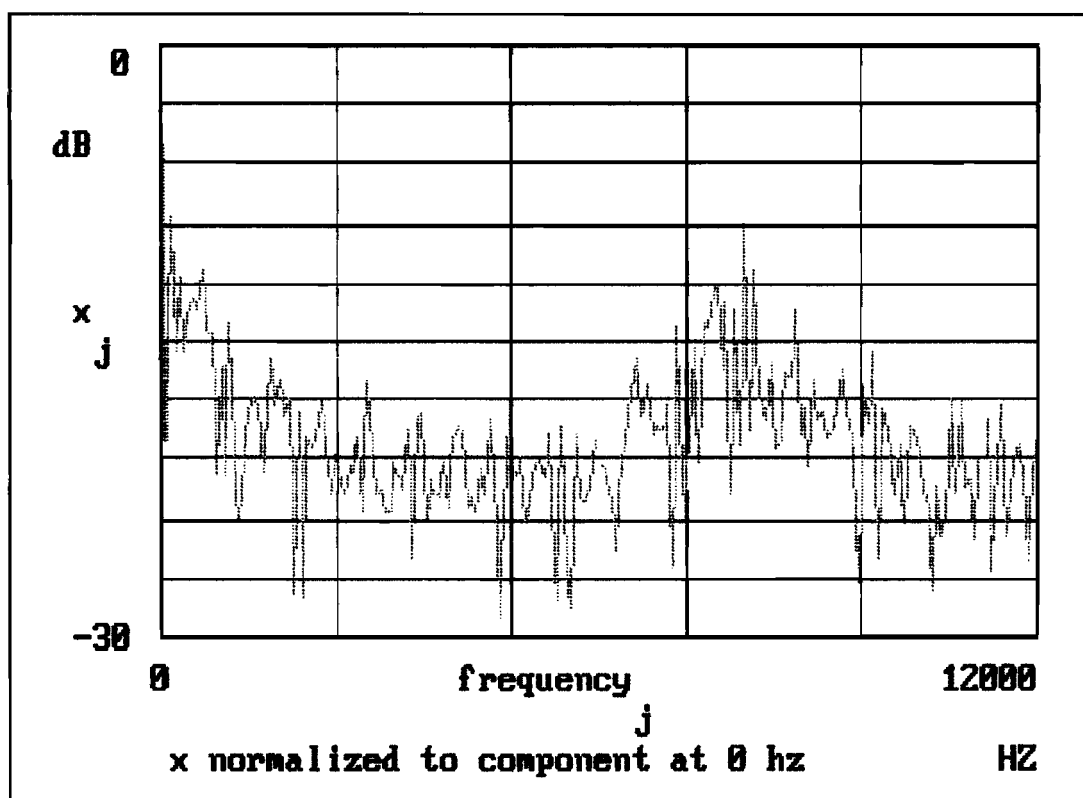


Figure 7- SPEECH SPECTRUM, SCALING ONLY

finding a method which will decrease the Peak to RMS ratio of the speech signal.

The choice of the method to reduce the Peak to Root Mean Square (RMS) ratio was complicated by the fact that the method used should not degrade the intelligibility or quality of the signal by any noticeable amount. J. C. R. Licklider found, in a study on the effects of amplitude distortion in speech signals, that if a speech signal is symmetrically peak clipped, there is essentially no reduction in intelligibility [15].

This method was used to clip the speech samples at approximately 30% of their maximum value. The peak clipping stage was followed by a gain stage which then boosted the amplitude of the clipped signals back up to the maximum. Because more of the samples have a  $\beta = 1$  after the clipping/gain stage this reduces the noise much further than simple scaling alone did. The results of the clipping/gain operation are shown in figures 8 and 9.

While the above operations did increase the signal amplitude and decrease the noise amplitude where a signal was actually present, it did nothing to help the periods at the beginning and end of the waveform where the signal is not present or has very little amplitude. To avoid having a noise "whistle" at the beginning and end of the speech, it was necessary to hand edit the speech data to cut out as much of the noise as possible while not removing too much of the actual speech. This was done by manipulating the amplitude



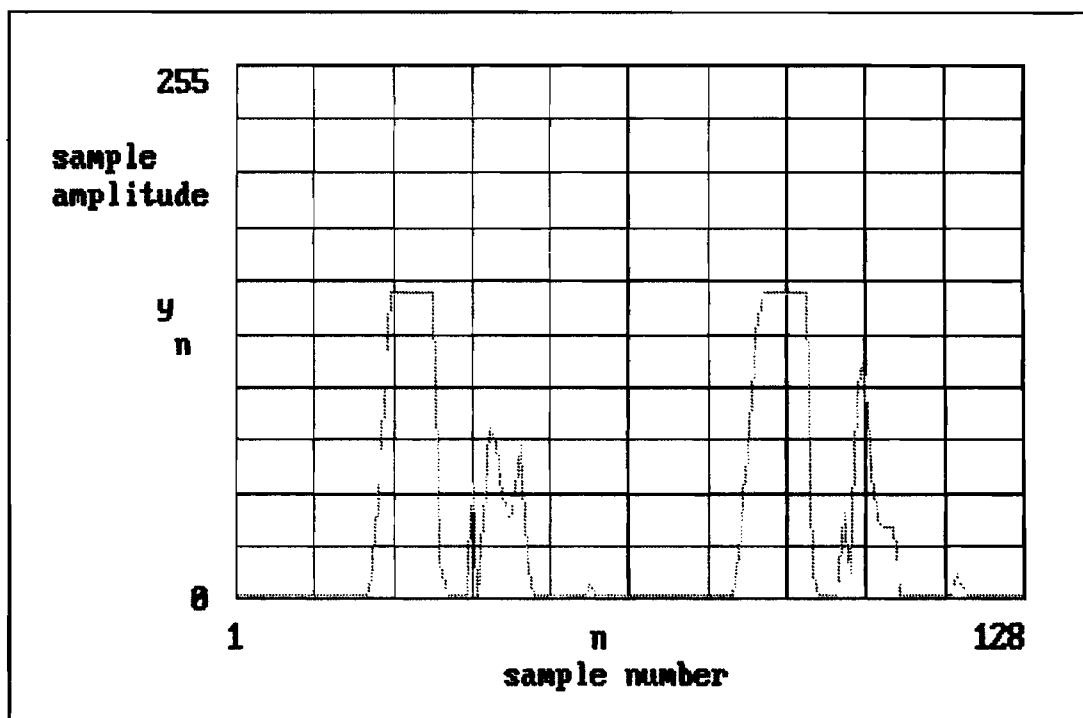


Figure 8- SPEECH SEGMENT, SCALING AND PEAK CLIPPING

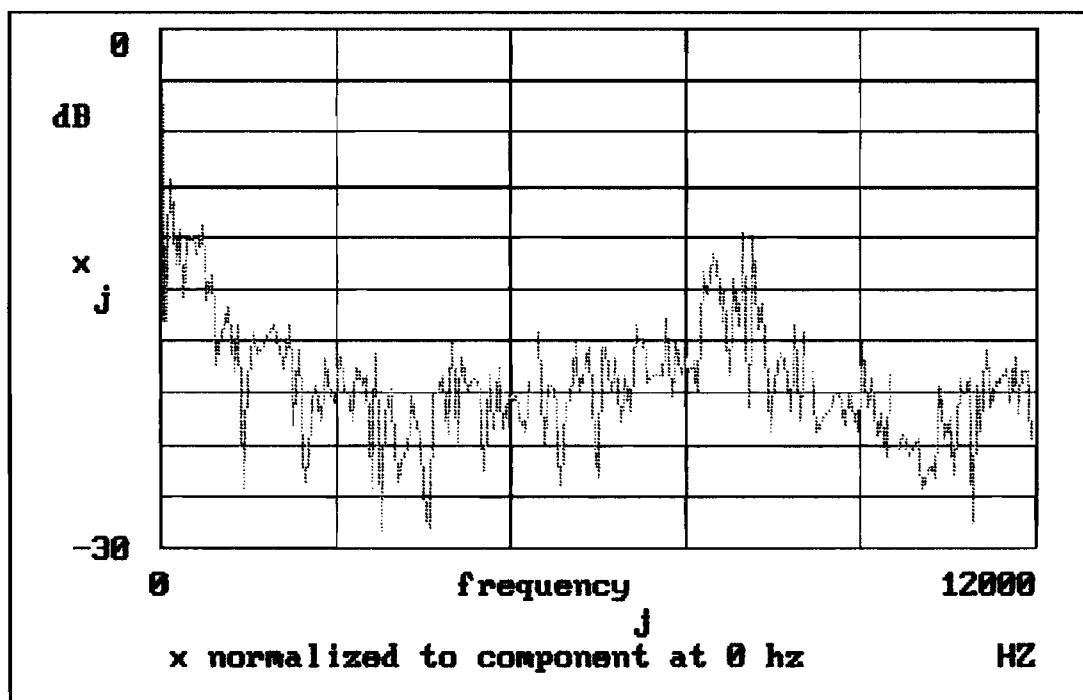


Figure 9- SPEECH SPECTRUM, SCALING AND PEAK CLIPPING

percentage and the amount of signal energy to retain parameters as described in section VI.

With the above additions to the speech synthesizer the output which it generated was intelligible and the quality was relatively high.

## VIII. COMPARISON TESTING

In order to assess the quality and intelligibility of the speech synthesizer a set of 10 sentences were recorded. The sentence are listed in Appendix E. Each sentence was composed of phonemes constructed specifically to test various characteristics of speech synthesizers [16]. In order to see how this speech synthesizer compares with what is currently available, the sentences were recorded with both the public domain synthesizer mentioned earlier and the waveform synthesizer described in this paper. The public domain synthesizer ran on an IBM PC XT while the waveform synthesizer ran on a PS 2 Model 50.

The test setup consisted of 2 tapes labeled A and B. Tape A contained the sentences as spoken by the public domain synthesizer and tape B contained the sentences spoken using the waveform synthesizer. Eight volunteers participated in the test. Each volunteer listened to each tape separately and answered questions about the speech. Sufficient dead time was allowed between sentences to allow completion of the questions.

For each sentence the volunteer was asked to do two things. First, the volunteer was asked to write down the words which were spoken in the sentence. The volunteer was told that words could be left blank if they could not understand the word. The second item the volunteer was asked to do was to subjectively rate the quality of the spoken sentence. The scale was from 1 to 5 with the instructions

stating that 1 was to correspond to a sentence which could not be understood and 5 corresponding to a sentence that was perfectly understandable.

The test results were divided into 2 parts. First a percentage of the words correctly identified for each synthesizer was determined. Second an average quality rating was determined. The percentage of words correctly identified can be used to judge the waveform synthesizer on its own merits while the subjective rating can be used to compare it to the public domain synthesizer.

The averaged results are as follows:

Tape A ( public domain ):	0.46	% correct words
	1.09	subjective rating
Tape B ( waveform ):	80.3	% correct words
	3.4	subjective rating

While these results show that the waveform synthesizer rating are better than the public domain synthesizer, it should be noted that the waveform synthesizer is running on a faster computer with a slightly different speaker interface (The Model 50 speaker is driven by a linear amplifier while the PC XT speaker is driven by a 75477 line driver). The public domain synthesizer will not run on the faster computers so a direct comparison is not possible, however these rating do show that the waveform synthesizer described in this paper is a significant improvement over synthesizers that are currently available.

## IX. CONCLUSIONS

This paper has presented a method to implement a speech synthesizer on personal computers which uses only the internal hardware available with no add-on boards or peripherals. Quality tests have been performed which showed that the output of the speech synthesizer has a good quality and the words are perfectly understandable approximately 80 % of the time.

The synthesizer is also commercially feasible. A vocabulary of 200 words will take up approximately 800 Kbytes of storage space. This is a reasonable figure considering the size of hard disk and high density floppy systems on present personal computers.

There are two basic areas where further work could be done to improve this synthesizer. The first is in the parameter optimization of the speech data. Currently, the clipping point, gain and the number of samples are determined by an iterative process while listening to the results. If a method could be developed which would automatically determine these parameters based on the data itself it could save a lot of time.

The second area where additional work could be done is in writing driver software to control the synthesizer. The driver software is the interface between an application program (say, a word processor) and the synthesizer. If driver software were developed it would be much easier for the application software to communicate with the synthesizer.

## BIBLIOGRAPHY

- [1] G. Heftman, "Speech Synthesizers- the Talk of the IC World". Electronic Design, March 5, 1981, p. 73.
- [2] L. Gunn, "Chips Try Teaching Computers to Speak, Listen". Electronic Design, May 25, 1989, pp. 33-36.
- [3] K. Fushikida, et. al., "A Text to Speech Synthesizer for the Personal Computer". IEEE Transactions on Consumer Electronics, Vol. CE-28, No. 3, August 1982, pp. 250-255.
- [4] W. D. Stanley, G. R. Dougherty, R. Dougherty, Digital Signal Processing. Reston Publishing, 1984.
- [5] A. A. Giordano and F. M. Hsu, Least Square Estimation with Applications to Digital Signal Processing. John Wiley & Sons, 1985.
- [6] J. Flanagan et al , "Speech Coding". IEEE Transactions on communications, VOL COM-27, No 4, April 1979, pp. 710-737.
- [7] IBM Corp., Personal System 2 Model 50 and 60 Technical Reference. IBM, 1987.
- [8] IBM Corp., DOS Technical Users Guide. 1986.

- [9] National Semiconductor Corp., Linear Databook 2. 1988.
  
- [10] Signetics Corp., High Speed CMOS Data Manual, 1986.
  
- [11] Interface Technologies, ML-16 Multi - lab Reference Manual, 1986.
  
- [12] Intel Corp., Micro System Components Handbook Volume II. 1985.
  
- [13] W. Tranter and R. Ziemer, Principles of Communications. Houghton Mifflon, 1985.
  
- [14] A. Oppenheim and R. Schafer, Digital Signal Processing. Prentice Hall Inc., Englewood Cliffs, NJ, 1975.
  
- [15] J. C. R. Licklider, "Effects of Amplitude Distortion upon the Intelligibility of Speech". Journal of the Acoustical Society of America, Vol. 18, No. 2, October, 1946, pp. 429-434.
  
- [16] A. W. F. Huggins and R. S. Nickerson, "Speech Quality Evaluation Using "Phoneme-Specific" Sentences". Journal Acoustic Society of America, Vol. 77 No. 5, May, 1985, pp. 1896-1906.

## VITA

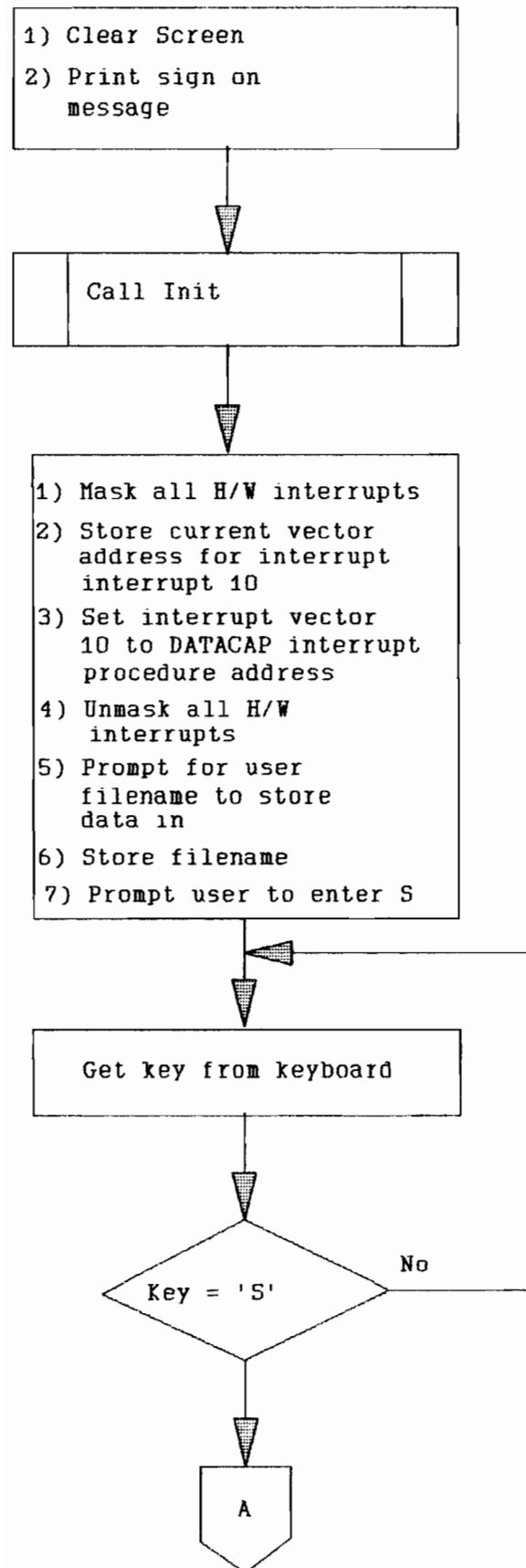
John Glenn Wallace II was born on April 1, 1961 in Dorton, KY. He received his primary and secondary education in Dorton, KY. His undergraduate studies consisted of one year at Pikeville College in Pikeville, KY and three years at the University of Kentucky. In 1983 he received a Bachelor of Science degree in Electrical Engineering from the University of Kentucky. He has been employed at McDonnell Aircraft company for 6 years, of which the last 4 have been spent in the Avionic Displays Laboratory.

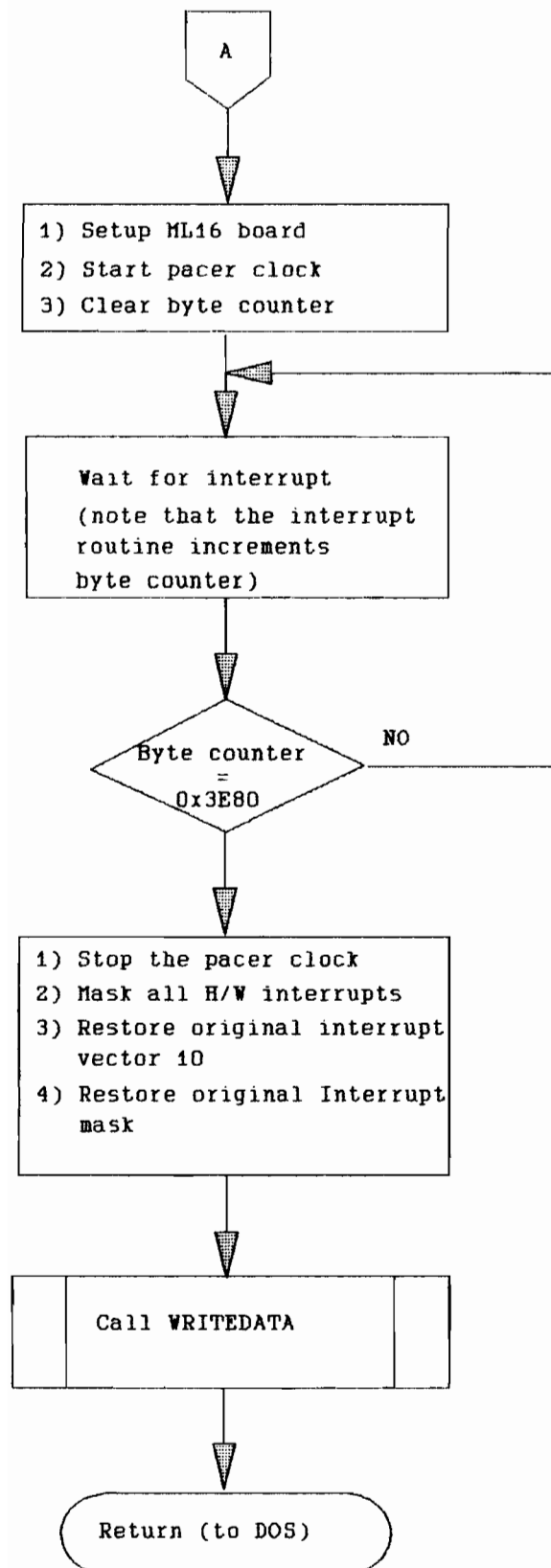
He has been enrolled in the Graduate School of Engineering at the University of Missouri- Rolla since 1984.



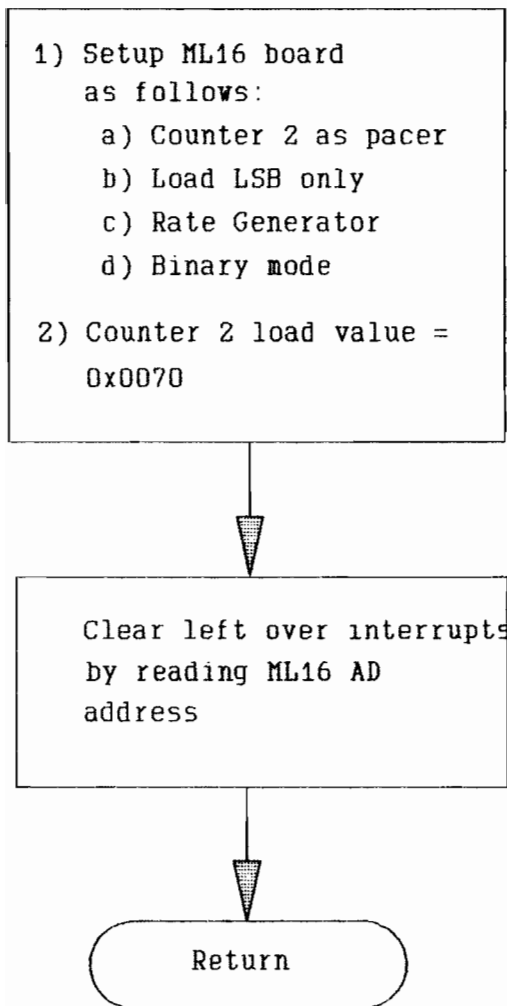
## APPENDIX A

### DATACAP FLOWCHART

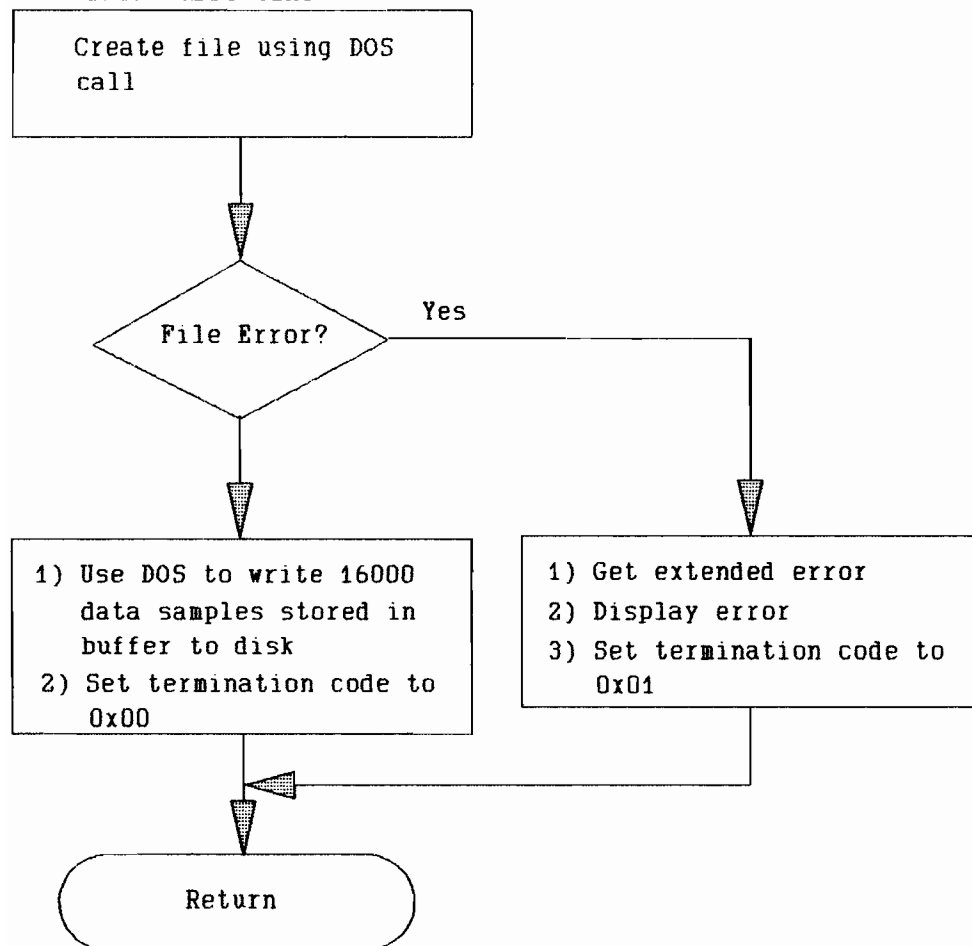




### INIT subroutine

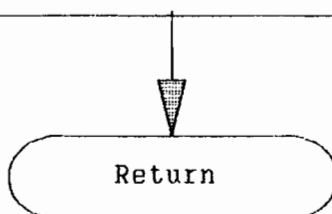


## WRITEDATA subroutine



## INTERRUPT routine

- 1) Mask H/W interrupts
- 2) Push registers and flags
- 3) Increment byte counter
- 4) Input sample from ML16
- 5) Store sample in buffer  
indexed by byte count
- 6) Issue an EOI to the PIC
- 7) Restore registers and  
flags
- 8) Unmask H/W interrupts



## APPENDIX B

### SPEAK INTERRUPT TIMING

The following is a listing of the interrupt procedure in the speak program. The number of 80286 clock cycles required for each instruction is given to the left side of the instruction.

```

Clk      Intrpt PROC
          ; Turn off interrupts and save status
3        CLI
3        PUSH DX
3        PUSH AX
3        PUSH BX
3        PUSH DI
3        PUSHF

          ; Output the data sample ( note that the data samples
          ; have been scaled so they that are not larger than
          ; 148d.

3        PUSH DS ; Push DS since this programs DS may not be
          ; the same as the data
5        MOV DS, Data_seg
3        MOV SI, CX
5        MOV AL, [SI] ; Address the buffer using the offset
          ; stored in CX
5        POP DS
2        INC CX ; Set offset to next byte
3        MOV DX, Port_8253_Timer_2
3        OUT DX, AL ; Output the value

          ; And increment the counter

7        INC Counter

Exit_intr:

          ; Increment the loop counter

3        MOV AX, Time_Counter
7        INC AX
3        MOV Time_Counter, AX

          ; And check to see if we've been here 440 times

3        CMP AX, 440
9        JZ Timer_Tick

          ; Otherwise
          ; Issue an EOI to the PIC

2        MOV AL, EOI
2        MOV DX, Int_0

```



```

3      OUT DX, AL

      Tidy_up:

5      POPF
5      POP DI
5      POP BX
5      POP AX
5      POP DX

2      STI
      ; Now return from the interrupt

25     IRET

      ; If we've been through the loop 440 times turn
      ; control over
      ; to the DOS timer interrupt handler to do its tasks.
      ; Notice that the IRET will be from the DOS Interrupt
      ; handler.

      Timer_tick:

3      MOV Time_Counter, 0
23     INT 60h
8      Last_addr:JMP Tidy_up

      intrpt ENDP

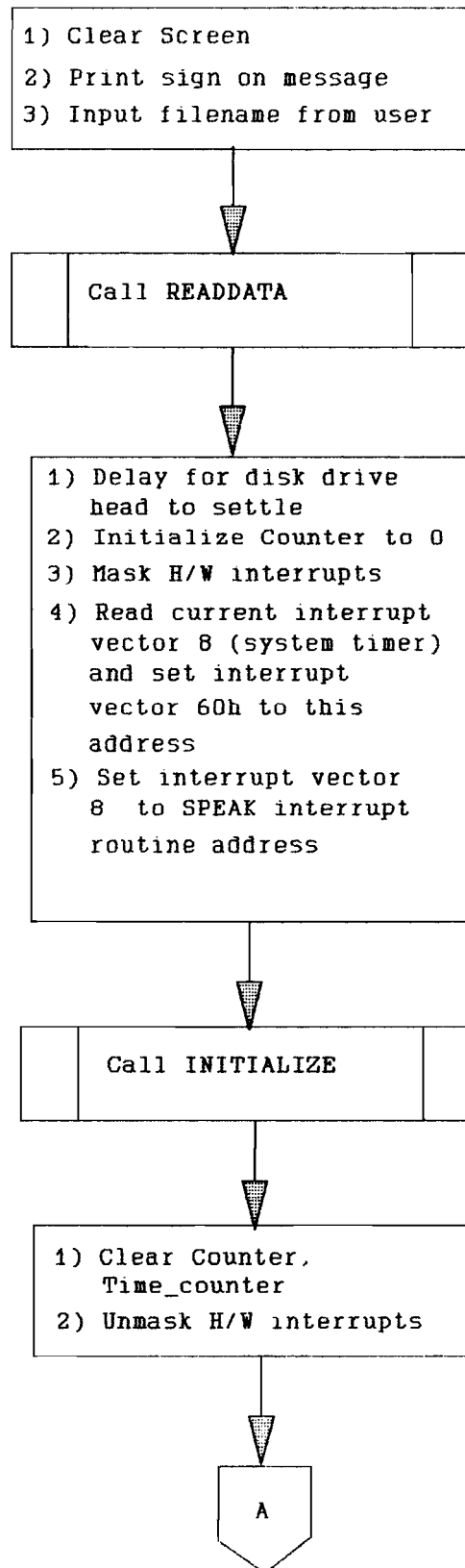
-----
      167 Instructions cycles in the speak interrupt procedure
+128 Instructions cycles in the BIOS interrupt procedure
-----
      295 Total instructions cycles

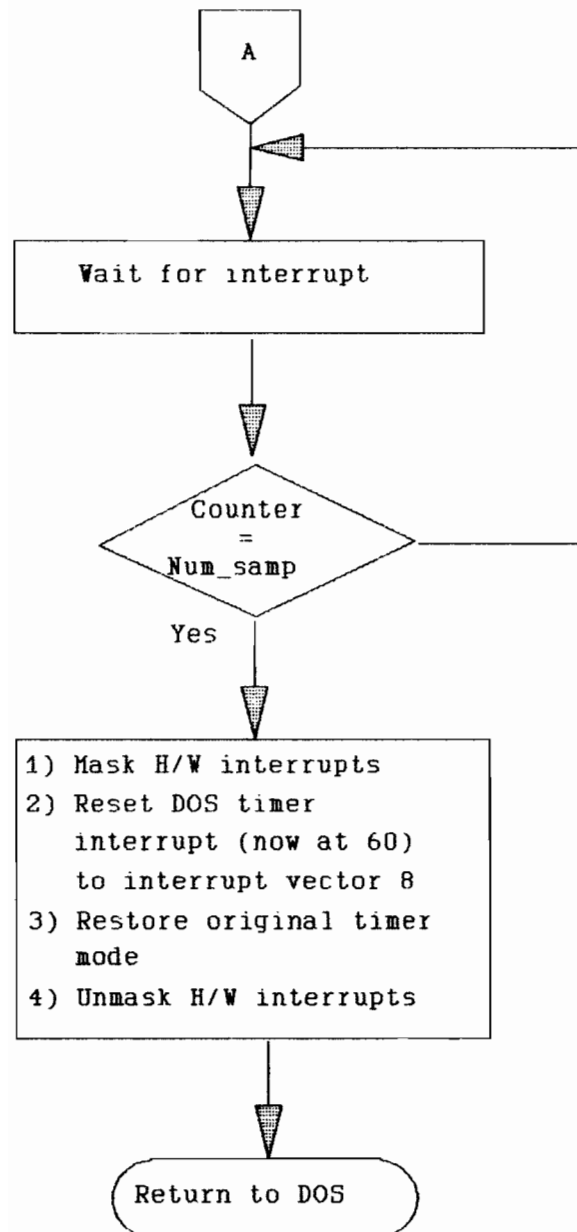
```

Clock rate = 295 clock cycles / 125  $\mu$ sec = 2.36 Mhz

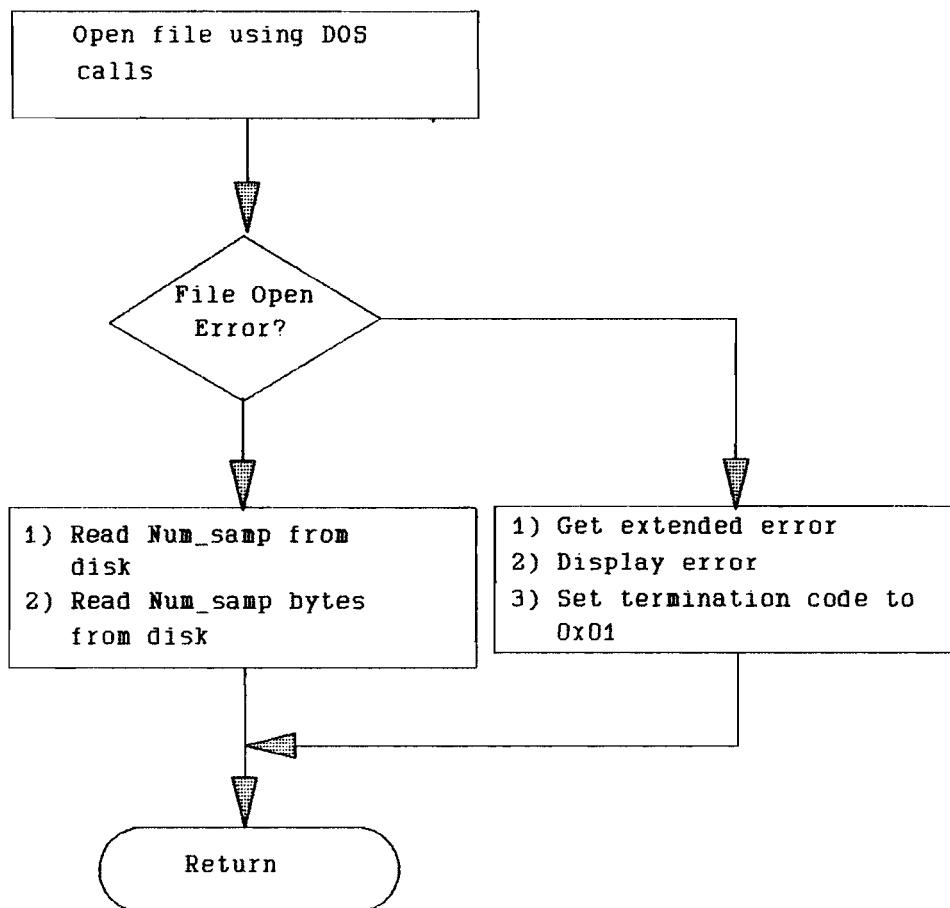
Note that the above number assumes no resident programs are stealing time during the bios timer interrupt and the processor is operating with no wait states inserted for memory or I/O operations.

APPENDIX C  
SPEAK FLOWCHART

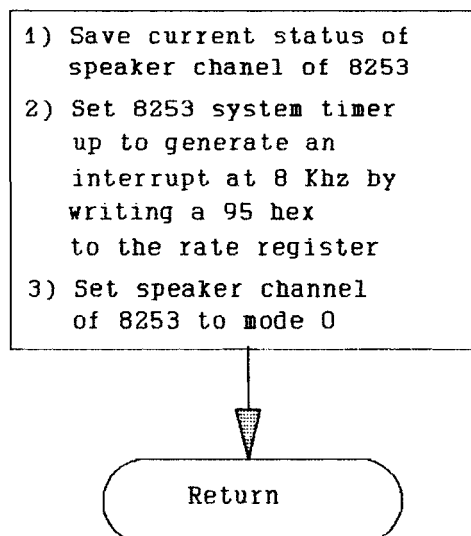




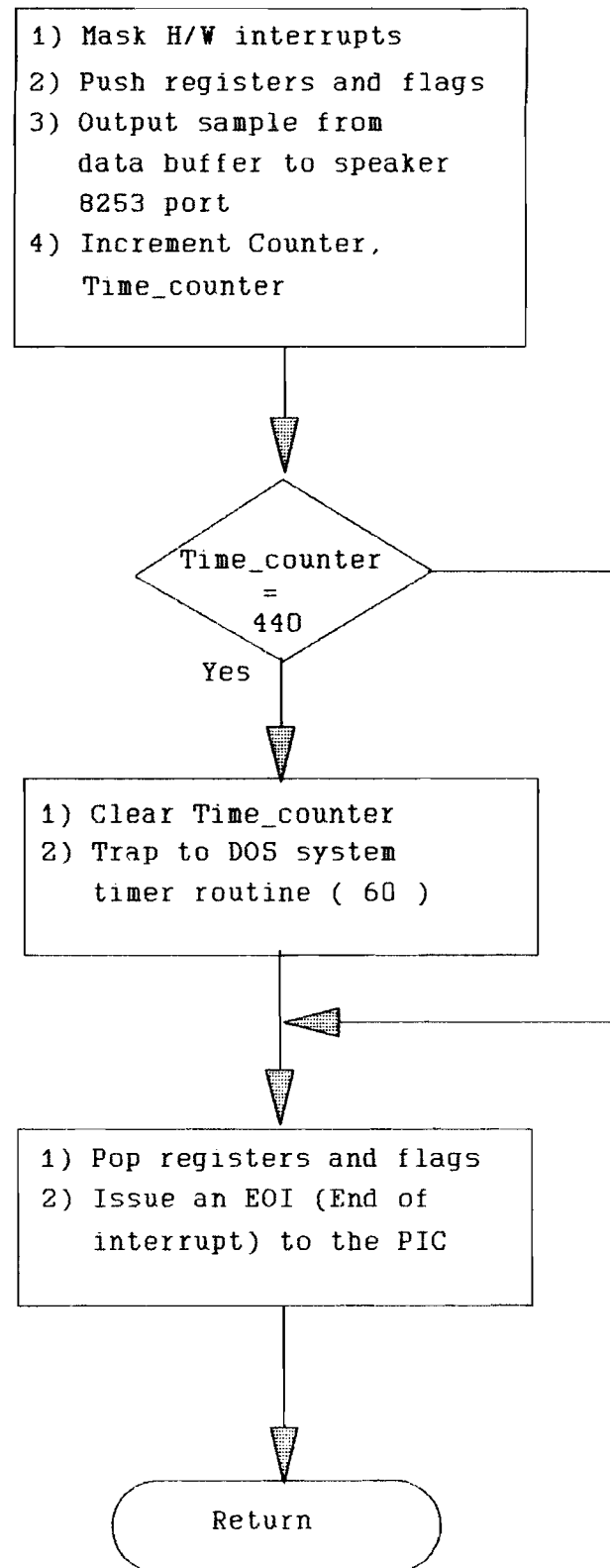
## READDATA subroutine



## INITIALIZE routine

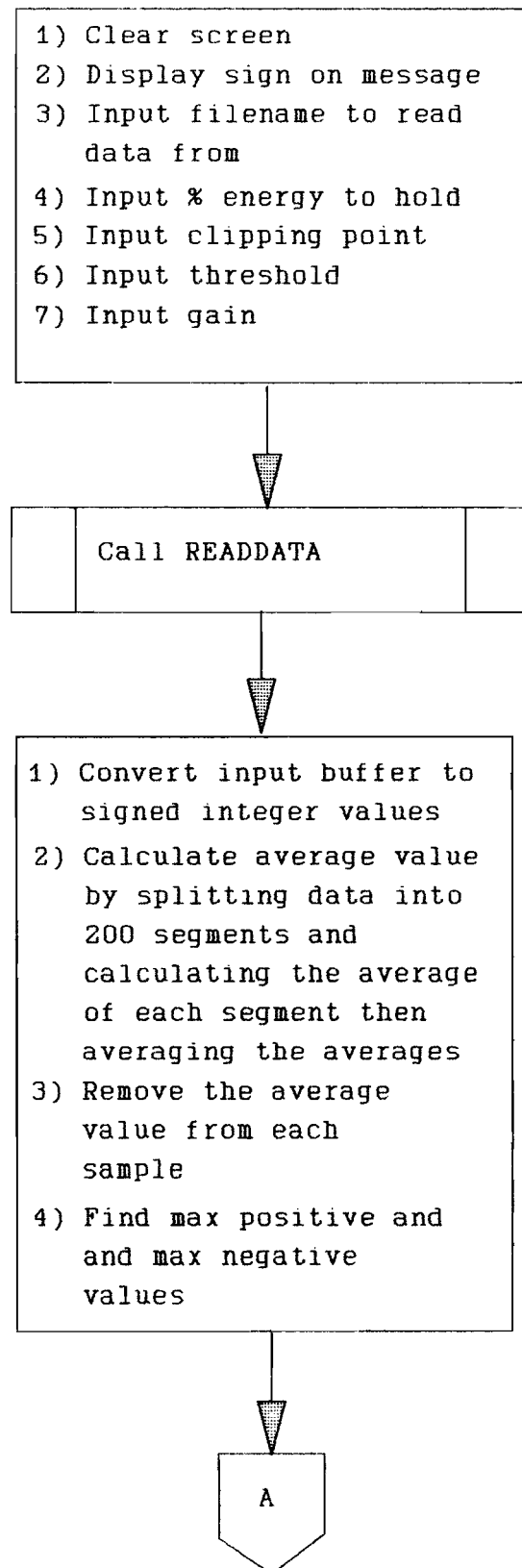


## INTERRUPT routine

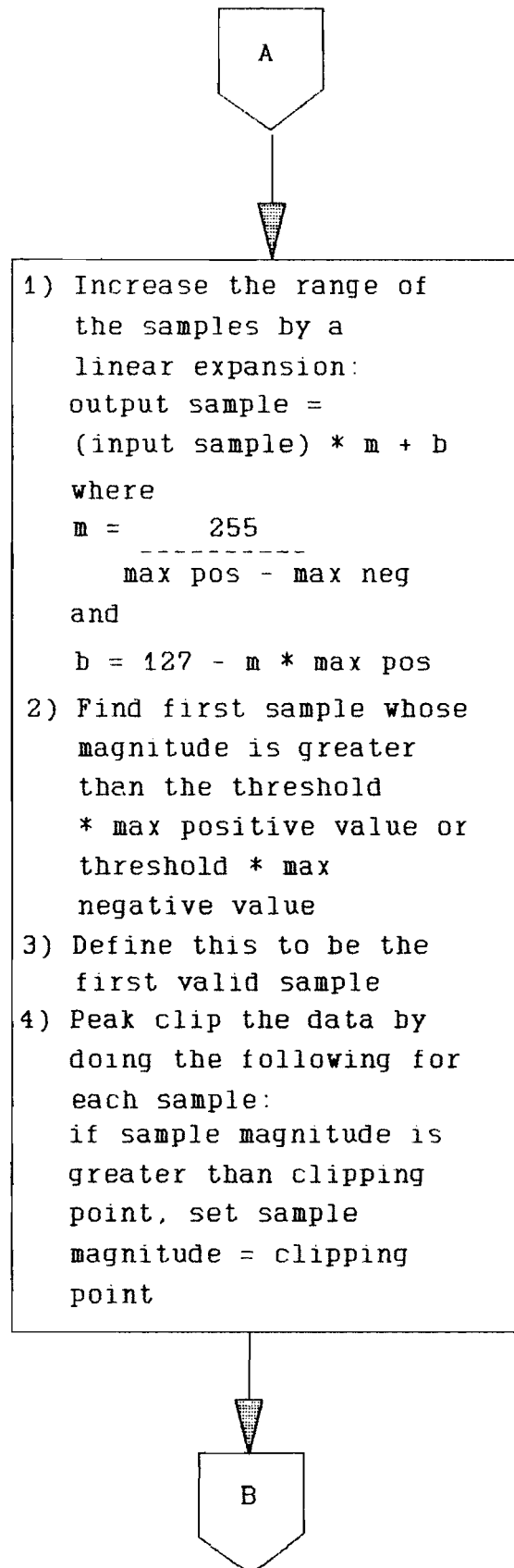


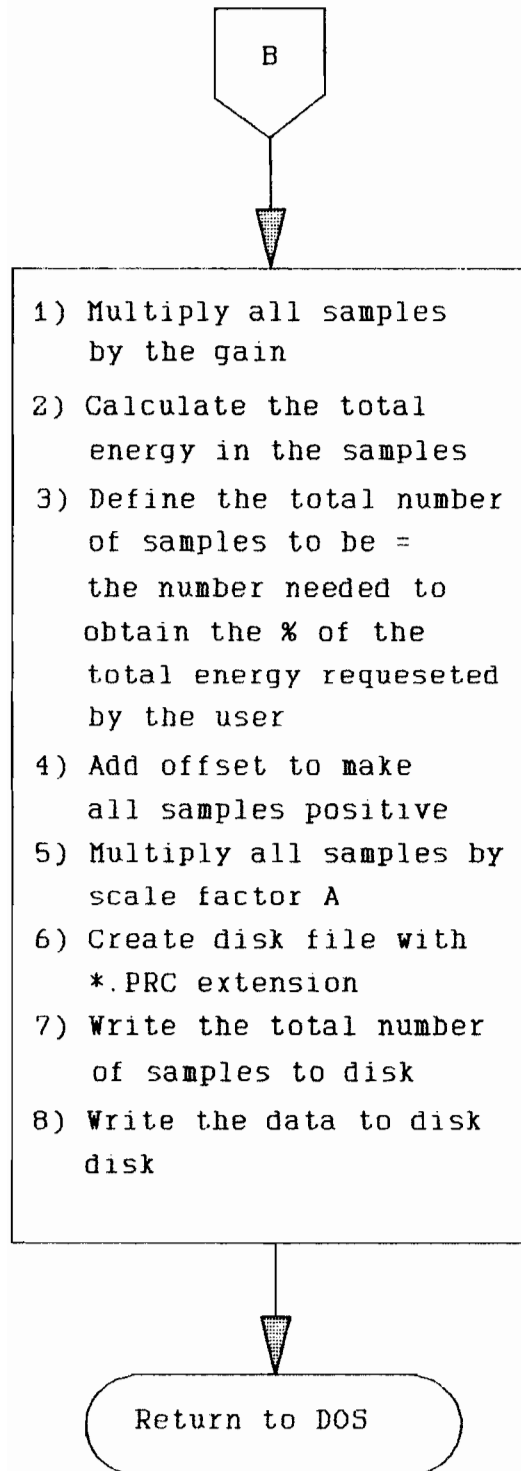
## APPENDIX D

### PREPRO FLOWCHART

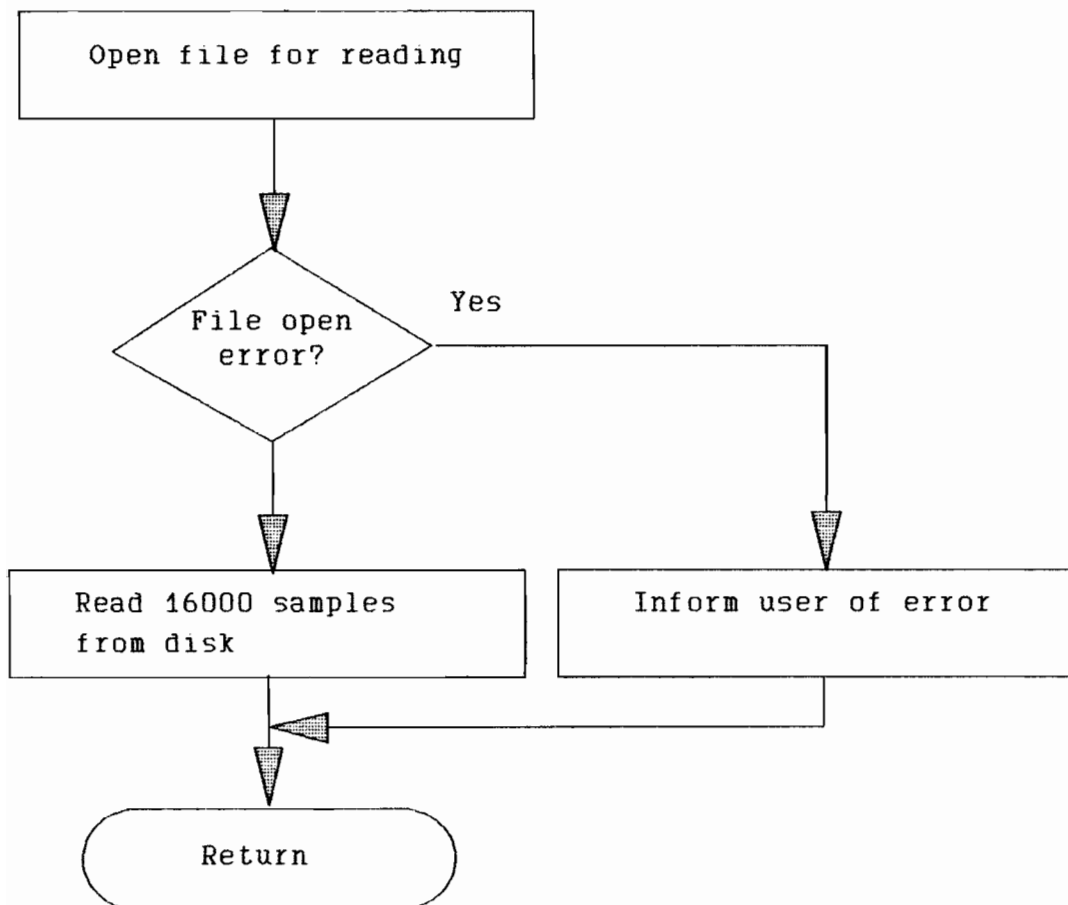








## READDATA subroutine



## APPENDIX E

### PHONEME SENTENCES

- 1) Why were you all weary?
- 2) Many young men owe money.
- 3) A normal animal will run away.
- 4) Grab a doggie bag.
- 5) Capture a cute puppy.
- 6) Those waves veer over.
- 7) A thief saw a fish.
- 8) Weave me a web above a poppy.
- 9) He steps off a path to cash a check.
- 10) Does John believe you were measuring the gun?